Jacobian neural network learning algorithm

甄景贤 (King-Yin Yan)

General.Intelligence@Gmail.com

The classical Back Prop algorithm is **error-driven**, but in many AI problems the "correct" answers are not given, instead the feedback is provided via **rewards**. When an answer is correct, the reward $R > 0$, error $\mathscr{E} = 0$; when the answer is incorrect, $R < 0$, but $\mathscr{E}$ is still unknown (because we don't know the correct answer). In other words, error-driven learning is inapplicable.

So I thought of a reward-driven learning method: assume the neural network maps $\boldsymbol{x}_0 \mapsto \boldsymbol{y}_0$, usually it also maps the neighborhood of $\boldsymbol{x}_0$ to the neighborhood of $\boldsymbol{y}_0$. If we wish to "strengthen" this pair of mapping, we can make a **bigger** neighborhood of $\boldsymbol{x}_0$ map to the same neighborhood close to $\boldsymbol{y}_0$. We will make this precise.

This type of learning algorithm should be very useful to AI, and currently the author is not aware of other alternatives for training [deep] neural networks via rewards.

A feed-forward neural network can be constructed this way:

$$\boldsymbol{y} = \boldsymbol{F}(\boldsymbol{x}) \tag{1}$$

$$\boldsymbol{y} = \oslash \overset{L}{W} \dots \oslash \overset{\ell}{W} \dots \oslash \overset{1}{W} \boldsymbol{x} \tag{2}$$

where W represents the matrix of **weights** on each layer ℓ.

(We shall use this later) The **inverse** of F is:

$$\boldsymbol{x} = \boldsymbol{F}^{-1}(\boldsymbol{y}) \tag{3}$$

$$\boldsymbol{x} = \overset{1}{\gtreqless} \obslash \dots \overset{\ell}{\gtreqless} \obslash \dots \overset{L}{\gtreqless} \obslash \, \boldsymbol{y} \tag{4}$$

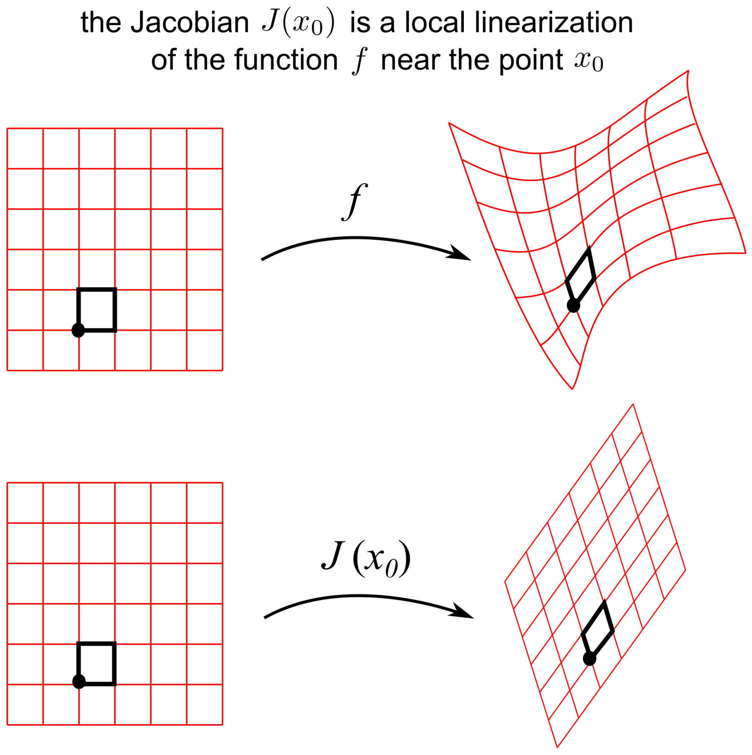
Note: $\gtreqless = W^{-1}$, $\obslash = \oslash^{-1}$, the shape is different.

Assume that in the space of $\boldsymbol{x}$ there is a volume element U, which transforms via $\boldsymbol{F}$ to a volume element V in the space of $\boldsymbol{y}$, then:

$$U = |J| \cdot V \tag{5}$$

$J = \left[\dfrac{\partial \boldsymbol{F}(x)}{\partial x} \right]$ is called the **Jacobian** matrix.

In our case, the value of $|J| = \left| \dfrac{\partial \boldsymbol{F}^{-1}(\boldsymbol{y})}{\partial \boldsymbol{y}} \right|$ at $\boldsymbol{y}_0$ represents "the change in volume from $\boldsymbol{y}_0 \mapsto \boldsymbol{x}_0$". (Below, we will see that the direction of $\boldsymbol{F}$ or $\boldsymbol{F}^{-1}$ is not too important, as either way the computational complexity is essentially the same.)


the Jacobian $J(x_0)$ is a local linearization
of the function f near the point x_0
f
$J(x_0)$

(6)

Every time we get a **positive reward**, we can let the Jacobian $|J|$ increase slightly:

$$|J| := \det \left[\frac{\partial \boldsymbol{F}^{-1}(\boldsymbol{y})}{\partial \boldsymbol{y}} \right]_{n \times n} \tag{7}$$

The subscript indicates that it is a $n \times n$ matrix.

The meaning of the Jacobian matrix is:

$$J = \left[\frac{\partial \text{ input}}{\partial \text{ output}} \right] \tag{8}$$

The neural network's input and output are both of dim n, so the Jacobian is naturally an $n \times n$ matrix.

To use **gradient descent**, we need to calculate these gradients: $\left[\frac{\partial |J|}{\partial \boldsymbol{W}} \right]$, their total number is the number of weights in the network $= \sum \ell \, \#(\overset{\ell}{W})$.

We need the formula for the derivative of the determinant:

$$\frac{d}{dt}|A(t)| = tr(\text{adj}(A) \cdot \frac{dA(t)}{dt}) \tag{9}$$
$$\text{adj}(A) := |A| \cdot A^{-1} \tag{10}$$

In other words, for each weight $w := \overset{\ell}{W}_{ij}$, we need to calculate:

$$\frac{\partial}{\partial w}|J| = tr(|J| \cdot J^{-1} \cdot \left[\frac{\partial J}{\partial w} \right]) \tag{11}$$

Note: $|J|$ and J^{-1} are functions of $\boldsymbol{y}_0$, we only need to calculate them once outside the big loop.

Now a problem arises in the calculation of $\left[\dfrac{\partial J}{\partial w}\right]_{n\times n}$:

$$\frac{\partial J}{\partial w} = \frac{\partial}{\partial w}\frac{\partial \boldsymbol{F}^{-1}}{\partial y} = \frac{\partial}{\partial w}\frac{\partial}{\partial \boldsymbol{y}} \; \overset{1}{\gtreqless}\oslash ... \overset{\ell}{\gtreqless}\oslash ... \overset{L}{\gtreqless}\oslash\, \boldsymbol{y} \tag{12}$$

This requires us to differentiation W^{-1} w.r.t. w; we can imagine the result would be very complicated. So we use a trick, by "reversing" the network, we use $\gtreqless$ to define the network weights, and then use $W = \gtreqless^{-1}$ during forward propagation.

The components of J are:

$$J_{ij} = \frac{\partial \boldsymbol{F}_i^{-1}}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\overset{1}{\gtreqless}\oslash ... \overset{\ell}{\gtreqless}\oslash ... \overset{L}{\gtreqless}\oslash\, \boldsymbol{y}\right]_i =: \nabla^1_{ij} \tag{13}$$

$$\begin{cases} \nabla^1_{ij} & := \displaystyle\sum_{k_1}[\,\overset{1}{\gtreqless}_{ik_1}\oslash'(y^2_{k_1})\nabla^2_{ij}\,] \\ \nabla^\ell_{ij} & := \displaystyle\sum_{k_\ell}[\,\overset{\ell}{\gtreqless}_{k_{\ell-1}k_\ell}\oslash'(y^{\ell+1}_{k_\ell})\nabla^{\ell+1}_{ij}\,] \\ \nabla^L_{ij} & := \qquad \overset{L}{\gtreqless}_{k_{L-1}j}\oslash'(y_j) \end{cases} \tag{14}$$

This situation is exactly analogous to the classical Back Prop algorithm; The above is just the application of the **chain rule**, with ∇^ℓ written separately for each layer, therefore ∇ is called the "local gradient". The above formula amounts to propagating the entire network one time, where every weight appears **exactly once**.

But our work is not finished yet; We need to calculate $\dfrac{\partial J_{ij}}{\partial \succeq} =: \dot{\nabla}^1_{ij}$.

(Let's define $\succeq := \overset{\ell}{\gtreqless}_{gh}$, $k_0 := i$, $k_L := j$)

Note: $\boldsymbol{x} = \boldsymbol{F}^{-1}(\boldsymbol{y})$, so $\boldsymbol{y}$ is the **independent variable**, $\succeq$ does not influence $\boldsymbol{y}$, so $\dfrac{\partial \boldsymbol{y}}{\partial \succeq} \equiv 0$.

There may be a problem here: can't figure out who depends on whom, so what follows may be wrong.

$\succeq$ would be an element of $\gtreqless$, but if $\succeq \notin \gtreqless$, all the terms below would vanish:

$$\begin{cases} \dot{\nabla}^1_{ij} = \displaystyle\sum_{k_1}\left[\overset{1}{\gtreqless}_{ik_1}\oslash'(y^2_{k_1})\dot{\nabla}^2_{ij}\right] \\ \dot{\nabla}^\ell_{ij} = \displaystyle\sum_{k_\ell}\left[\overset{\ell}{\gtreqless}_{k_{\ell-1}k_\ell}\oslash'(y^{\ell+1}_{k_\ell})\dot{\nabla}^{\ell+1}_{ij}\right] \\ \dot{\nabla}^L_{ij} = \qquad \overset{L}{\gtreqless}_{k_{L-1}j}\oslash'(y_j) \equiv 0 \end{cases} \tag{15}$$

So what is left over is just this term:

$$\frac{\partial J_{ij}}{\partial \succeq} = \sum_{k_1}\left[\overset{1}{\gtreqless}_{k_0k_1}\oslash'(y^2_{k_1})...\sum_{k_{\ell-2}}\left[\overset{\ell-2}{\gtreqless}_{k_{\ell-3}k_{\ell-2}}\oslash'(y^{\ell-1}_{k_{\ell-2}})...\right.\right. \tag{16}$$

$$\begin{cases} ...\;\overset{\ell-1}{\gtreqless}_{k_{\ell-2},g}\oslash'(y^\ell_g)\oslash'(y^{\ell+1}_h)\nabla^{\ell+1}_{ij}\Big]\Big] \\ ...\;\overset{\ell-1}{\gtreqless}_{k_{\ell-2},g}\oslash'(y^\ell_g)\oslash'(y^{\ell+1}_h)\Big]\Big] \qquad \text{if } \succeq \in \text{last layer} \end{cases}$$

The above formula means: For each layer we repeat a block of $\left[\sum \gtrsim \oslash'\right]$, until we encounter $\gtrsim = \overset{\ell}{\gtrsim}_{gh}$, then we replace with the terminal form.

In contrast to classical Back Prop, the above formula gives us only one element in an $n \times n$ matrix; From the complexity point of view, calculating the ∇ for each weight is at least n^2 times as costly as classical Back Prop (even though we may re-use some intermediate computation results). Recall that $n = \dim \boxed{\text{state space}}$.

We can understand it thusly: For each weight we try to calculate its influence towards the Jacobian, but the Jacobian is a **global** property of the network. The key seems to lie in how each weight **influences** the Jacobian.

Now let's look back at this higher-level formula:

$$\frac{\partial}{\partial w}|J| = tr(|J| \cdot J^{-1} \cdot \left[\frac{\partial J}{\partial w}\right]) \tag{17}$$

$$= |J| \cdot tr(\left[\frac{\partial y}{\partial x}\right] \cdot \left[\frac{\partial}{\partial w}\frac{\partial x}{\partial y}\right]) \tag{18}$$

$$= |J| \cdot \sum_{ij} \left(\frac{\partial y}{\partial x}\right)_{ij} \left(\frac{\partial}{\partial w}\frac{\partial x}{\partial y}\right)_{ij} \tag{19}$$

The most critical (slowest) part is the $(i,j) \in n \times n$ summation. The first factor inside $\sum$ is the Jacobian J, the second factor is the $\nabla_w J^{-1}$ that we just calculated.

Back Prop's ∇ has the form $\dfrac{\partial \boxed{\text{output}}}{\partial \boxed{\text{weights}}}$

whereas our ∇ has the form $\left[\dfrac{\partial}{\partial \boxed{\text{weights}}} \dfrac{\partial \boxed{\text{input}}}{\partial \boxed{\text{output}}}\right]_{n\times n}$.

In fact we just need to calculate the **approximate** direction and size of $\nabla_w|J|$. Currently in our code we use this trick: ignore the smaller terms in (**??**), so we don't need to do all of n^2 products.

Or perhaps we can regard $|J|(\gtrsim)$ as a function of the weight $\gtrsim$, and then use its Taylor series expansion to approximate?

Notes

- Parameters are organized hierarchically ("deep")
- Jacobian of F at x in terms of W is easy to calculate
-

References